Theoretical Computer Science 75 (1990) 67-83
North-Holland

A DECLARATIVE SEMANTICS OF FLAT GUARDED HORN CLAUSES FOR PROGRAMS WITH PERPETUAL PROCESSES

Masaki MURAKAMI*

Institute for New Generation Computer Technology, 1-4-28 Mita, Minato-ku, Tokyo 108, Japan

Abstract. A declarative semantics of a concurrent programming language based on Horn logic such as Flat GHC is presented. The domain of input/output (I/O) histories is introduced. The model of a program is defined as a set of I/O histories. The notion of truth is redefined for goal clauses and sets of guarded clauses. The semantics of a program is defined as the maximum model of the program. We also show that the semantics is characterized as the greatest fixpoint of the function obtained from the program. The properties of programs that contain perpetual computation controlled by guard-commit mechanisms can be discussed using the semantics.

1. Introduction

In recent years, several concurrent programming languages based on Horn logic have been investigated. Examples are PARLOG [3], Concurrent Prolog [19] and GHC [23]. In such languages, the notion of processes which execute infinite computations controlled by guard-commit mechanisms communicating with other processes using input/output streams can be represented naturally. Several results on the formal semantics of these languages are reported [10, 15, 16, 17, 22, 20, 21]. However, these results are based on the operational approach. Thus, they should be considered as a formal specification of the language processing system. In order to give a logical base for program verification methods or transformation methods, a kind of declarative semantics is expected.

In pure Horn logic programming languages, the result for declarative semantics based on the least fixpoint is reported in [1, 9]. In this approach, the denotation of a program is given as the minimum model of the set of Horn clauses, in other words, the set of unit clauses which is equivalent to the program. The set of unit clauses is characterized as the least fixpoint of the function obtained from the set of definite clauses. In this approach, we can characterize the set of solutions of the program independently from the execution mechanisms. This approach is one of the best ways of appreciating the clarity of logic programs. Extensions of this approach for programs which contain infinite computations are also reported in [9, 14].

However, these results are reported for pure Horn logic languages. They cannot be applied to parallel languages which contain the notion of a guard-commit

*Present address: Fujitsu Limited, International Institute for Advanced Study of Social Information Science, 17-25, Shinkamata, 1-chome, Ota-ku, Tokyo 144, Japan.

mechanism directory. A model theory must be reconstructed for Horn logic with the commit operator. Thus several extensions are reported for such languages [7, 8]. Levi [8] discusses the semantics of Flat GHC programs as the sets of guarded atoms. A guarded atom is a guarded clause such that all atoms in the guard part and the body part are unifications. For example,

$$p(X_1, \ldots, X_n, Y_1, \ldots, Y_m) :- X_1 = \tau_1, \ldots, X_n = \tau_n \mid Y_1 = \tau'_1, \ldots, Y_m = \tau'_m.$$

It can be considered as a unit clause of Flat GHC.

However, in this approach, the guarded atom describes only the relation between the input substitutions and the compute substitutions which are obtained when the goal succeeds. It is difficult to discuss the infinite computation of the program only with such relation. As Takeuchi [21] reported, there are two programs which cannot be distinguished only by relations of input and output substitutions, and output different results when they are executed in parallel with other processes. Thus, the information for the intermediate result of the computation is necessary to discuss the semantics of such programs.

This paper introduces a new declarative semantics for Flat GHC programs which contain perpetual processes. The notion of *input/output* (*I/O*) *history* is introduced instead of the notion of the guarded atom. Intuitively an I/O history denotes an example of a computation path of a program which is generated when the program is executed without any failure or deadlock. We define the notion that a goal clause or a set of guarded clauses is true with respect to a set of I/O histories. The semantics of a program is defined as the maximum set of I/O histories which makes the program true, in other words, the maximum model of the program.

The notion of a true goal clause with respect to the model of a program does not necessarily mean that the goal clause succeeds on the program. That is, not only all successful goal clauses are true but also goals which do not succeed finitely but can be executed infinitely without failure or deadlock are true. A goal clause with a goal which suspends can also be true if the goal can be activated with some input from other processes.

This paper also shows that the semantics of a program can be characterized as the greatest fixpoint of the function obtained from the program.

2. Guarded stream

This section introduces the notion of the *guarded stream*. For simplicity we only consider programs on the domain of lists of {**a**, **b**}.

Definition 2.1. Let *Var* be a set of variables, *Fun* = {**a**, **b**, **nil**, **cons**} be a set of function symbols. Each element of *Fun* has its arity. The arity of **a**, **b** and **nil** is 0, and the arity of **cons** is 2.

Definition 2.2. Let *Terms* be the set of terms defined as follows:
(1) if $\tau \in Var$ or $\tau \in \{\mathbf{a}, \mathbf{b}, \mathbf{nil}\}$, then $\tau \in Terms$;
(2) if $\tau_1, \tau_2 \in Terms$, then $\mathbf{cons}(\tau_1, \tau_2) \in Terms$.

Definition 2.3. A term τ is said to be *simple*, when $\tau \in Var$, $\tau \in \{\mathbf{a}, \mathbf{b}, \mathbf{nil}\}$ or τ has the form of $\mathbf{cons}(X, Y)$, and X and Y are different variables.

Definition 2.4. A mapping, $\sigma: Var \rightarrow Terms$ is called a *substitution* if it satisfies the following condition: if $\Sigma = \{X \mid \sigma X \neq X, X \in Var\}$, then Σ is a finite set.

We expand the domain of substitutions from *Var* to *Terms*.

Definition 2.5. For each $\tau, \tau_1, \tau_2 \in Terms$, $\sigma\tau$ is recursively defined as follows:

$$\sigma\tau = \begin{cases} \sigma X & \text{if } \tau \text{ is } X \in Var, \\ \tau & \text{if } \tau \in \{\mathbf{a}, \mathbf{b}, \mathbf{nil}\}, \\ \mathbf{cons}(\sigma\tau_1, \sigma\tau_2) & \text{if } \tau = \mathbf{cons}(\tau_1, \tau_2). \end{cases}$$

$\mathbf{cons}(X, Y)$ is denoted $[X \mid Y]$ and **nil** is denoted [ ].

Definition 2.6. Let τ be a simple term and $X \in Var$. $X = \tau$ is a *simple substitution form* or a *substitution form* simply. $X = X$ is denoted *true*.

A substitution form is intended to represent an execution of unification goal in a program. The substitution computed during the computation is obtained by the set of unification goals that is executed during the computation. Thus, a finite set $\sigma = \{X_1 = \tau_1, \ldots, X_n = \tau_n\}$ of substitution forms, where each X_j is not identical to τ_j for $1 \leqslant j \leqslant n$, is intended to represent a substitution that is obtained by executing unification goals: $X_1 = \tau_1, \ldots, X_n = \tau_n$ in appropriate order. (Of course any set of substitution forms does not always define a substitution.) For example, a substitution represented by the set $\{\mathbf{X} = [\mathbf{A} \mid \mathbf{Y}], \mathbf{A} = \mathbf{a}\}$ maps **X** to $[\mathbf{a} \mid \mathbf{Y}]$. This idea can be considered as a special case of the notion of *the set of constraints* in [18]. We identify a finite set of substitution forms with the substitution if there is no confusion, in the rest of this paper.

Definition 2.7. Let σ be a set of substitution forms that defines the substitution and U be a substitution form. If $\sigma \cup \{U\}$ defines a substitution, then U is *consistent* with σ. Furthermore, $\sigma \cup \{U\}$ defines the same substitution to σ, then σ *implies* U, denoted $\sigma \models U$. We denote $\sigma \not\models U$ if σ does not imply U.

Definition 2.8. Let σ be a set of simple substitution forms. If σ is a substitution or equal to $\bigcup_{k \rightarrow \infty} \theta_k$ for a sequence of substitution θ_k such that

$$\theta_0 = \emptyset, \qquad \theta_{k+1} = \theta_k \cup \{X = \tau\}$$

where $X = \tau$ is consistent with θ_k and $\theta_k \not\models (X = \tau)$, then σ is an *ω-substitution.* Intuitively an ω-substitution defines a mapping from a term to an infinite term.

Definition 2.9. Let V be a set of variables ($V \subset Var$), and Σ be the set defined from a mapping σ as Definition 2.4. If $\Sigma \subset V$, then σ is *restricted to V.* If $\Sigma \cap V = \emptyset$, then σ is *invariant on V.*

The notion of I/O history introduced in this paper corresponds to the notion of the unit clause for pure Horn logic programs. I/O history is an extension or modification of a guarded atom of [8]. An I/O history is denoted as follows with the head part H, which denotes a form of a process, and the body part GU, which denotes a trace of inputs and outputs of the process

$$H :- GU.$$

H is defined in the next section. GU is a set of tuples $\langle \sigma | U_b \rangle$ where σ is a substitution which is expressed in the form of a set of simple substitution forms, and U_b is a formula which represents an execution of a unification in the body part of some clause. Intuitively, $\langle \sigma | U_b \rangle$ means that the arguments of the process are instantiated with σ, then unification U_b is executed. For instance, in the following program:

$$\mathbf{p1(X, Y) :- X = [A|X1],\ \ A = a | Y = [B|Y1],\ \ B = b,\ p1(X1, Y1).}$$

$$\mathbf{p1(X, Y) :- X = [B|X1],\ \ B = b | Y = [A|Y1],\ \ A = a,\ p1(X1, Y1).}$$

The following is an example of I/O history which denotes the computation such that **p1** reads **a** in input stream **X** first, writes **b** in output stream **Y**, then reads **b** and writes **a**.

$$\begin{aligned}\mathbf{p1(X, Y) :-}\ & \{\langle \{\mathbf{X = [A|X1], A = a}\} | \mathbf{Y = [B|Y1]}\rangle,\\ & \langle \{\mathbf{X = [A|X1], A = a}\} | \mathbf{B = b}\rangle,\\ & \langle \{\mathbf{X = [A|X1], A = a, X1 = [B1|X2], B1 = b}\} | \mathbf{Y1 = [A1|Y2]}\rangle,\\ & \langle \{\mathbf{X = [A|X1], A = a, X1 = [B1|X2], B1 = b}\} | \mathbf{A1 = a}\rangle, \ldots\}.\end{aligned}$$

An I/O history of a process H represents a possible execution of the process. Thus, there exist different I/O histories for different executions which commit to different clauses. There may be different I/O histories for different scheduling.

The body part of an I/O history represents a normal execution of Flat GHC programs, thus GU is well founded with the partial order of execution, namely, for any $\langle \sigma_1 | U_{b1} \rangle, \langle \sigma_2 | U_{b2} \rangle \in GU$, if $\sigma_1 \subset \sigma_2$, then U_{b1} is executable before U_{b2}.

In the rest of this section, the notion of guarded stream is introduced which characterizes the normal executions of GHC programs.

Definition 2.10. Let τ be a simple term and $X \in Var$. $X ?= \tau$ is a *simple test form* or a *test form* simply.

Definition 2.11. For a substitution σ, $X \in Var$, and a simple term τ, $\langle \sigma \mid uni(X, \tau) \rangle$ is *a guarded unification*, where $uni(X, \tau)$ denotes $X = \tau$ or $X ?= \tau$. σ is the *guard part* of $\langle \sigma \mid uni(X, \tau) \rangle$ and $uni(X, \tau)$ is the *active part.*

Intuitively, if $uni(X, \tau)$ is a substitution form, it denotes a unification which actually instantiates X, and if it is a test form, it denotes a test unification.

Definition 2.12. Let $\langle \sigma \mid U \rangle$ be a guarded unification. $|\langle \sigma \mid U \rangle|$ is the set of substitution forms or test form defined as

$$|\langle \sigma \mid U \rangle| = \{U\} \cup \sigma.$$

Definition 2.13. Let GU be a set of guarded unifications. For $\langle \sigma_1 \mid u_1 \rangle, \langle \sigma_2 \mid u_2 \rangle \in GU$,

$$\langle \sigma_1 \mid u_1 \rangle \prec \langle \sigma_2 \mid u_2 \rangle$$

holds if and only if there exists a substitution θ_1 such that $\theta_1 \sigma_1 = \sigma_2$ and there is no substitution θ_2 such that $\sigma_1 = \theta_2 \sigma_2$.

It is easy to show that $\prec$ is a well founded ordering.

Definition 2.14. Let GU be a set of guarded unifications and Gu be a finite subset of GU. Gu is *closed from below* iff for any $gu, gu' \in GU$ such that $gu' \prec gu$, if $gu \in Gu$ then $gu' \in Gu$. Let Gu be a finite subset of GU that is closed from below. A guarded unification $gu \in GU$ is an *upper bound of* Gu, if for any $gu' \in Gu$, $gu \not\prec gu'$ and $gu' \neq gu$. An upper bound gu of Gu is a *minimal* upper bound if there is no $gu'' \in GU$ such that gu'' is an upper bound of Gu and $gu'' \prec gu$.

We denote the set of substitution forms (that is representing the substitution), $\{X = \tau \mid \langle \sigma \mid uni(X, \tau) \rangle$ or $\langle \{\ldots, X = \tau, \ldots\} \mid U \rangle$ is in $Gu\}$ as $|Gu|$.

Definition 2.15. A set of guarded unifications GU is a *guarded stream* if the following are true. For any finite subset Gu that is closed from below and any minimal upper bound $\langle \sigma \mid U \rangle$ of Gu,

(1) if U is a substitution form, then U is consistent with the substitution $|Gu| \cup \sigma$, and $|Gu| \cup \sigma \not\models U$.

(2) If U is a test form, then U is consistent with the substitution $|Gu| \cup \sigma$. If $|GU| \cup \sigma \models U$, then U is a test form.

(3) For any $U' \in \sigma$, U' is consistent with the substitution $|Gu|$, and there is no substitution θ such that $|Gu| = \theta\sigma$.

Example 2.16. The following are examples of guarded streams. GU_1 represents an execution of the process which takes a sequence of "a"s in Z and outputs a sequence of "b"s in Y. GU_2 represents an execution of the process which takes a sequence of "a"s in X and a sequence of "b"s in Y respectively, and merges them and outputs the result to Z.

$$GU_1 = \{gu_{1i}(1 \leq i)\},$$

$$gu_{11} = \langle \{\mathbf{Z} = [\mathbf{A}|\mathbf{Z1}], \mathbf{A} = \mathbf{a}\} | \mathbf{Y} = [\mathbf{B}|\mathbf{Y1}] \rangle$$

$$gu_{12} = \langle \{\mathbf{Z} = [\mathbf{A}|\mathbf{Z1}], \mathbf{A} = \mathbf{a}\} | \mathbf{B} = \mathbf{b} \rangle$$

$$gu_{13} = \langle \{\mathbf{Z} = [\mathbf{A}|\mathbf{Z1}], \mathbf{A} = \mathbf{a}, \mathbf{Z1} = [\mathbf{A1}|\mathbf{Z2}], \mathbf{A1} = \mathbf{a}\} | \mathbf{Y1} = [\mathbf{B1}|\mathbf{Y2}] \rangle$$

$$gu_{14} = \langle \{\mathbf{Z} = [\mathbf{A}|\mathbf{Z1}], \mathbf{A} = \mathbf{a}, \mathbf{Z1} = [\mathbf{A1}|\mathbf{Z2}], \mathbf{A1} = \mathbf{a}\} | \mathbf{B1} = \mathbf{b} \rangle$$

$$\vdots$$

$$GU_2 = \{gu_{2j}(1 \leq j)\}$$

$$gu_{21} = \langle \{\mathbf{X} = [\mathbf{A0}|\mathbf{X1}], \mathbf{A0} = \mathbf{a}\} | \mathbf{Z} = [\mathbf{A}|\mathbf{Z1}] \rangle$$

$$gu_{22} = \langle \{\mathbf{X} = [\mathbf{A0}|\mathbf{X1}], \mathbf{A0} = \mathbf{a}\} | \mathbf{A} = \mathbf{a} \rangle$$

$$gu_{23} = \langle \{\mathbf{X} = [\mathbf{A0}|\mathbf{X1}], \mathbf{A0} = \mathbf{a}, \mathbf{Y} = [\mathbf{B}|\mathbf{Y1}], \mathbf{B} = \mathbf{b}\} | \mathbf{Z1} = [\mathbf{A1}|\mathbf{Z2}] \rangle$$

$$gu_{24} = \langle \{\mathbf{X} = [\mathbf{A0}|\mathbf{X1}], \mathbf{A0} = \mathbf{a}, \mathbf{Y} = [\mathbf{B}|\mathbf{Y1}], \mathbf{B} = \mathbf{b}\} | \mathbf{A1} = \mathbf{b} \rangle$$

$$gu_{25} = \langle \{\mathbf{X} = [\mathbf{A0}|\mathbf{X1}], \mathbf{A0} = \mathbf{a}, \mathbf{Y} = [\mathbf{B}|\mathbf{Y1}], \mathbf{B} = \mathbf{b}, \mathbf{Y1} = [\mathbf{B1}|\mathbf{Y2}], \mathbf{B1} = \mathbf{b}\} | \mathbf{Z2} = [\mathbf{B2}|\mathbf{Z3}] \rangle$$

$$gu_{26} = \langle \{\mathbf{X} = [\mathbf{A0}|\mathbf{X1}], \mathbf{A0} = \mathbf{a}, \mathbf{Y} = [\mathbf{B}|\mathbf{Y1}], \mathbf{B} = \mathbf{b}, \mathbf{Y1} = [\mathbf{B1}|\mathbf{Y2}], \mathbf{B1} = \mathbf{b}\} | \mathbf{B2} = \mathbf{b} \rangle$$

$$\vdots$$

The following notion is defined to obtain the guarded stream representing the computation of a goal clause from the guarded streams which represent the computation of each goal in the goal clause.

Definition 2.17. Let $GU_1, \ldots, GU_n$ be guarded streams, and $Gu_k (1 \leq k)$ be as follows:

$$\begin{aligned} Gu_0 = \{\langle \sigma | U \rangle | \exists i, \langle \sigma | U \rangle \in GU_i, \\ \forall U' \in \sigma, ((\exists Gu \subset \bigcup_j GU_j, |Gu| \models U') \Rightarrow \\ (|Gu| - \{X = \tau | \langle \sigma'' | X = \tau \rangle \in Gu\} \models U') \\ )\} \end{aligned}$$

$$
\begin{aligned}
Gu_{k+1} = Gu_k \cup \\
\{\langle \sigma | U\rangle | \exists \sigma'((\exists i, \langle \sigma' | U\rangle \in GU_i) \wedge \\
(\forall U' \in \sigma'(\exists Gu \subset Gu_k . |Gu| \models U') \vee \\
(\forall Gu \subset \bigcup_j GU_j, \\
(|Gu| \models U' \Rightarrow \\
(|Gu| - \{X = \tau | \langle \sigma'' | X = \tau\rangle\} \models U') \\
) \\
) \\
) \wedge \\
(\sigma = (\sigma' - \{U' | |Gu_k| \models U'\}) \cup \\
\{U'' | \exists Gu \subset Gu_k \\
(\exists U' \in \sigma'((|Gu| \models U') \vee \\
(\forall \theta \subseteq |Gu|, |Gu| \neq \theta \Rightarrow \theta \not\models U') \\
) \\
) \wedge \\
(U'' \in \sigma'' \text{ for some } \langle \sigma'' | U''\rangle \in Gu) \\
\} \\
) \\
) \\
\}
\end{aligned}
$$

and let GU be

$$GU = \bigcup_{k \to \infty} Gu_k.$$

If GU is a guarded stream and if

$$\{U | \langle \sigma | U\rangle \in GU\} = \{U | \exists i \langle \sigma | U\rangle \in GU_i\}$$

then GU is a *synchronized merge of* $GU_1, \ldots, GU_n$, and is denoted

$$GU_1 \| \cdots \| GU_n.$$

If $n = 1$, then the synchronized merge can always be defined and it is equal to GU_1 itself.

Informally, the Gu_k can be understood as follows. Let $\langle \sigma | U\rangle \in GU_i$; this means that when σ is provided from the environment of $GU_1 \| \cdots \| GU_n$, U can be executed after at most k rounds of internal communication among GU_j. For instance, in Definition 2.17, for $\langle \sigma | U\rangle \in GU_i$, if $\langle \sigma | U\rangle \in Gu_1$ then that means U waits σ from outside of $GU_1 \| \cdots \| GU_n$ and waits nothing from $GU_j (j \neq i)$. If $\langle \sigma | U\rangle \in Gu_{k+1}$ then that means U waits some inputs from outside of $GU_1 \| \cdots \| GU_n$ and waits that $Ub_1, \ldots, Ub_m$ is executed in some GU_j such that it is already found that it waits σ_h $(1 \leq h \leq m)$ from outside. If Ub_h waits $U' \in \sigma_h$ from outside of $GU_1 \| \cdots \| GU_n$, then U also waits U'.

Consider the following guarded streams:

$$GU_1 = \{\langle\{\mathbf{X}=\mathbf{a}\}\,|\,\mathbf{Y}=\mathbf{b}\rangle\},$$

$$GU_2 = \{\langle\{\mathbf{Y}=\mathbf{b}\}\,|\,\mathbf{X}=\mathbf{a}\rangle\}.$$

They represent computations of the **g1(X, Y)** and **g2(Y, X)** where

$$\mathbf{g1(X, Y)} :- \mathbf{X}=\mathbf{a}\,|\,\mathbf{Y}=\mathbf{b},$$

$$\mathbf{g2(Y, X)} :- \mathbf{Y}=\mathbf{b}\,|\,\mathbf{X}=\mathbf{a}.$$

In this case,

$$\{U\,|\,\langle\sigma\,|\,U\rangle \in GU\} = \emptyset.$$

On the other hand,

$$\{U\,|\,\exists i\langle\sigma\,|\,U\rangle \in GU_i\} = \{\mathbf{X}=\mathbf{a}, \mathbf{Y}=\mathbf{b}\}.$$

Thus $GU_1 \,\|\, GU_2$ cannot be defined. It is impossible to obtain the guarded stream which represents the computation of goal clause **g1(X, Y), g2(Y, X)** from GU_1 and GU_2. In fact, neither **X** nor **Y** is instantiated by execution of **g1(X, Y), g2(Y, X)**.

Example 2.18. Let GU_1 and GU_2 be the same as Example 2.16.

$$Gu_1 = \{gu_{21}, gu_{22}\},$$

$$Gu_2 = Gu_1 \cup \{\langle\{\mathbf{X}=[\mathbf{A0}\,|\,\mathbf{X1}], \mathbf{A0}=\mathbf{a}\}\,|\,\mathbf{Y}=[\mathbf{B}\,|\,\mathbf{Y1}]\rangle,$$

$$\langle\{\mathbf{X}=[\mathbf{A0}\,|\,\mathbf{X1}], \mathbf{A0}=\mathbf{a}\}\,|\,\mathbf{B}=\mathbf{b}\rangle\}$$

$$Gu_3 = Gu_2 \cup$$

$$\{\langle\{\mathbf{X}=[\mathbf{A0}\,|\,\mathbf{X1}], \mathbf{A0}=\mathbf{a}\}\,|\,\mathbf{Z1}=[\mathbf{A1}\,|\,\mathbf{Z2}]\rangle,$$

$$\langle\{\mathbf{X}=[\mathbf{A0}\,|\,\mathbf{X1}], \mathbf{A0}=\mathbf{a}\}\,|\,\mathbf{A1}=\mathbf{b}\rangle\}$$

$$Gu_4 = Gu_3 \cup$$

$$\{\langle\{\mathbf{X}=[\mathbf{A0}\,|\,\mathbf{X1}], \mathbf{A0}=\mathbf{a}, \mathbf{A1}=\mathbf{a}\}\,|\,\mathbf{Y1}=[\mathbf{B1}\,|\,\mathbf{Y2}]\rangle,$$

$$\langle\{\mathbf{X}=[\mathbf{A0}\,|\,\mathbf{X1}], \mathbf{A0}=\mathbf{a}, \mathbf{A1}=\mathbf{a}\}\,|\,\mathbf{B1}=\mathbf{b}\rangle\}$$

$$\vdots$$

A1 is expected to be instantiated to **a** in the passive part of an element of Gu_4, however, it is instantiated to **b** in the active part of an element of Gu_3. Thus, $\bigcup_{k\to\infty} Gu_k$ is not a guarded stream. Therefore, the synchronized merge of GU_1 and GU_2 cannot be defined.

3. Model theoretic semantics

This section introduces notions which correspond to the Herbrand base and unit clauses, for parallel logic language based on the notion of guarded streams. First, a parallel language based on Horn logic is presented. The language is essentially a subset of Flat GHC [23] with only one system predicate, =: unification of a variable term and a simple term. Furthermore all clauses are assumed to be in a *normal form*, namely all arguments in the head part are different variable terms. However it is not difficult to show that the language presented here does not lose any generality compared to Flat GHC using the modified version of the transformation algorithm for the strong normal form [22].

Let *Pred* be a finite set of predicate symbols. Each element of *Pred* has its arity. We denote each element of *Pred* using lower-case letters.

Definitions 3.1. Let $H, B_1, B_2, \ldots, B_n$ be an atomic formula defined from *Pred*, every term which appears in the argument of H be a different variable, and $U_{g1}, \ldots, U_{gm}$ and $U_{b1}, \ldots, U_{bh}$ be simple substitution forms. The following is a *guarded clause.*

$$H :- U_{g1}, \ldots, U_{gm} | U_{b1}, \ldots, U_{bh}, B_1, B_2, \ldots, B_n.$$

A finite set of guarded clauses is a *program.* We define $Var(H) = \{X_1, X_2, \ldots, X_k\}$ when H is $p(X_1, X_2, \ldots, X_k)$.

Definition 3.2. Let p be an element of *Pred* with arity k, $X_1, X_2, \ldots, X_k$ be different variables and σ be an ω-substitution. Then $\sigma p(X_1, X_2, \ldots, X_k)$ is a *goal.*

Definition 3.3. A sequence of goals $g_1, \ldots, g_n$ is a *goal clause.*

Definition 3.4. Let GU be a guarded stream and V be a finite set of variables. The *restriction of GU by V*, $GU \downarrow V$ is the set defined as

$$GU \downarrow V = \{\langle \sigma | uni(X, \tau)\rangle | \langle \sigma | uni(X, \tau)\rangle \in GU, X \in V_k \text{ for some } k\}$$

where

$$V_0 = V$$

$$V_{i+1} = V_i \cup$$

$$\{X | \exists gu \in GU, \exists uni(Y, \tau) \in |gu|, X \text{ appears in } \tau, Y \in V_i \text{ and}$$

$$\forall gu' \in GU, \text{ if } gu' < gu, \text{ then } X \text{ does not occur in } gu'\}.$$

If GU is a guarded stream then $GU \downarrow V$ is also a guarded stream.

Definition 3.5. For a guarded stream GU and an atom $p(X_1, X_2, \ldots, X_k)$, a *pseudo I/O history* t is

$$p(X_1, X_2, \ldots, X_k) :- GU$$

where $p \in Pred$ with arity k, $X_1, X_2, \ldots, X_k$ are different variables, and

$$GU \downarrow Var(p(X_1, X_2, \ldots, X_k)) = GU.$$

$p(X_1, X_2, \ldots, X_k)$ is called the *head part of* t and GU is called the *body part of* t. Intuitively, GU only contains variables which are *visible* from outside through the head part.

In pseudo I/O history, the same computation can be represented in several ways. In other words, if t_1 and t_2 are identical except for the names of variables which do not appear in the head parts, they are considered to represent the same computation. Thus an equivalent relation is introduced to the domain of pseudo I/O histories.

Definition 3.6. A mapping $\sigma: Var \to Var$ is a *renaming mapping* if there exists a mapping σ', such that $\sigma\sigma' = \sigma'\sigma$.

Let GU be a guarded stream and σ be a renaming mapping. σGU is a guarded stream, defined as

$$\sigma GU = \{\sigma gu \mid gu \in GU\}$$

where

$$\sigma gu = \langle \sigma * \theta \mid uni(\sigma Y, \sigma\tau') \rangle,$$

for $gu = \langle \theta \mid uni(Y, \tau') \rangle$ and $\sigma * \theta$ is the substitution defined as

$$\sigma * \theta = \{\sigma X = \sigma\tau \mid X = \tau \in \theta\}.$$

It is easy to show that if GU is a guarded stream, then σGU is also a guarded stream.

Definition 3.7. Let $t_1: H :- GU_1$ and $t_2: H :- GU_2$ be pseudo I/O histories with the same head part H. If there exists a renaming mapping $\sigma: GU_1 \to GU_2$ invariant on $Var(H)$ such that $\sigma GU_1 = GU_2$ then $t_1 \approx t_2$ holds.

It is easy to show that $\approx$ is an equivalent relation. We denote the quotient set of all pseudo I/O histories with $\approx$ as *I/O-hist*. Each element of *I/O-hist* is called an *I/O history*.

Definition 3.8. An *interpretation* is any subset of *I/O-hist*.

Definition 3.9. Let t be an I/O history and g be a goal. $H :- GU$ is a *trace of* g if the following hold.

(1) There exists an ω-substitution σ such that $\sigma H = g$.

(2) For any $\langle \theta \mid U \rangle \in GU$, $\theta \subset \sigma$.

(3) For any $\langle \theta \mid U \rangle \in GU$, if U is a substitution form $X = \tau$, then σ does not instantiate X, and if U is a test form then $\sigma X = \sigma\tau$.

A mapping σ does not instantiate a variable X if $\sigma X = Y (\in Var)$ and there is no Z such that $\sigma Z = Y$ except X.

Definition 3.10. Let I be a interpretation and g be a goal. g is *true* on I if there exists a trace of $\sigma g \in I$ for some ω-substitution, σ.

For goals $g_1, \ldots, g_n$, let $t_1, \ldots, t_n$ be their traces. If they are I/O histories that are obtained when these goals are executed in parallel, the shared variables in t_i and t_j must have some value, and they occur as subterms of values of the same variables in t_i and t_j. The following notion is introduced to formalize these conditions.

Definition 3.11. $t_1, \ldots, t_n$ is *variable compatible* if for any i and j $(1 \leqslant i, j \leqslant n)$, the set of variables which occur in both t_i and t_j is equivalent to $Common(t_i, t_j)$ defined as follows.

$$COM_0(t_i, t_j) = Var(H_i) \cap Var(H_j)$$

$$\begin{aligned} COM_{k+1}(t_i, t_j) = COM_k(t_i, t_j) \cup \\ \{X \mid \exists Y \in COM_k(t_i, t_j), \\ \exists \tau, \text{ which has the form of } X, [X \mid Z], \text{ or } [Z \mid X], \\ \exists gu_i \in GU_i, ((Y = \tau) \in |gu_i| \vee (Y\ ?{=}\ \tau) \in |gu_i|) \wedge \\ \forall gu_i' \in GU_i, \text{ if } gu_i' \prec gu_i \\ \text{then } X \text{ does not occur in any } U \in |gu_i'| \wedge \\ \exists gu_j \in GU_j, ((Y = \tau) \in |gu_j| \vee (Y\ ?{=}\ \tau) \in |gu_j|) \wedge \\ \forall gu_j' \in GU_j, \text{ if } gu_j' \prec gu_j \\ \text{then } X \text{ does not occur in any } U \in |gu_j'|.\} \end{aligned}$$

$$Common(t_i, t_j) = \bigcup_{k \to \infty} COM_k$$

where H_m denotes the head part of t_m and GU_m denotes the body part. Obviously, for $n = 1$, t_1 is variable compatible.

Definition 3.12. Let I be an interpretation and $g_1, \ldots, g_n$ be a goal clause. $g_1, \ldots, g_n$ is *true* on I if there exists a trace $t_i \in I$ for every σg_i $(1 \leqslant i \leqslant n)$ for some ω-substitution σ, and there exists a synchronized merge $GU_1 \| \cdots \| GU_n$ where $GU_1, \ldots, GU_n$ are body parts of elements of $t_1, \ldots, t_n$, which are variable compatible.

The empty goal clause is always true.

It is easy to show by the following proposition that Definition 3.11 is well defined, namely the truth value of $g_1, \ldots, g_n$ does not depend on which element is selected from the trace of each goal.

Proposition 3.13. *Let $GU_1, \ldots, GU_n$ be guarded streams and σ be a renaming mapping. If there exists $GU_1 \| \cdots \| GU_n$, then there also exists $\sigma GU_1 \| \cdots \| \sigma GU_n$.*

The proof is straightforward from the definition of synchronized merge.

For a given goal g, it is easy to show that g is true if and only if the goal clause with only one goal g is true.

Definition 3.14. Let GU be a guarded stream and θ_1, θ_2 be sets of simple substitution forms. The set $GU \bowtie (\theta_1, \theta_2)$ is defined as follows if it is a guarded stream.

$$GU \bowtie (\theta_1, \theta_2) = \{\langle \sigma \mid U_b \rangle \mid \langle \sigma' \mid U'_b \rangle \in GU, \sigma = \sigma' \cup \theta_1 - \theta_2,$$
$$\text{If } (\theta_1 \cup \theta_2) \not\models U_b, \text{ then } U'_b = U_b,$$
$$\text{and if } (U_b \text{ is } X = \tau) \wedge (\theta_1 \cup \theta_2) \models U_b$$
$$\text{then } U'_b \text{ is } X \ ?= \tau\}.$$

Definition 3.15. Let D be a finite set of guarded clauses and I be an interpretation. *I is a model of D* if for any $t \in I$, there exists a clause $H :- U_{g1}, \ldots, U_{gm} \mid X_1 = \tau_1, \ldots, X_h = \tau_h, B_1, \ldots, B_k \in D$, each element of t has the following form:

$$H :- \{\langle \{U_{g1}, \ldots, U_{gm}\} \mid uni(X_1, \tau_1) \rangle, \ldots, \langle \{U_{g1}, \ldots, U_{gm},\} \mid uni(X_h, \tau_h) \rangle\} \cup$$
$$((GU_1 \| \cdots \| GU_k) \bowtie (\{U_{g1}, \ldots, U_{gm}\}, \{X_1 = \tau_1, \ldots, X_h = \tau_h\})) \downarrow Var(H)$$

where GU_i is the body part of some instance of a trace $t_i (\in I)$ of the goal σB_i for some ω-substitution $\sigma = \{U_{g1}, \ldots, U_{gm}\} \cup \{X_1 = \tau_1, \ldots, X_h = \tau_h\} \cup \sigma'$, which is restricted to $Var(H) \cup \{X_1, \ldots, X_n\}$ and

$$\forall \langle \theta \mid U \rangle \in (GU_1 \| \cdots \| GU_k) \bowtie (\{U_{g1}, \ldots, U_{gm}\}, \{X_1 = \tau_1, \ldots, X_h = \tau_h\}), \quad \theta \subset \sigma.$$

The following proposition is easy to show from the definition of models.

Proposition 3.16. *Let $M_i (i \in Ind)$ be a class of models of D for a set of indices Ind. Then, $\bigcup_{i \in Ind} M_i$ is also a model of D.*

From Proposition 3.16, it is easy to show that there exists a unique maximum model for a given D. The semantics of D is defined as the maximum model of D. A goal clause $g_1, \ldots, g_m$ is true on D if it is true on the maximum model of D. Intuitively, the maximum model is the set of all computations without failure or deadlock on D.

The maximum model is defined for characterizing the goal clauses which run *normally* without failure or deadlock on the program as true on the model of the program. However *goal clauses which run normally* are not necessarily the successful goals. That is goals which run infinitely are regarded as goals that run normally. Furthermore a goal clause suspending goal can also be true. Consider the following example:

p(X, Y) :- X = a | Y = b.

q(X, Y) :- Y = b | X = a.

t(X, Y) :- X = a | true, p(X, Y), q(X, Y).

Although goal **t(X, Y)** suspends on this program, if **X** is instantiated to **a**, then the execution proceeds. The maximum fixpoint of this program contains:

t(X, Y) :- {⟨{X = a} | true⟩, ⟨{X = a} | Y = b⟩, ⟨{X = a} | X ?= a⟩}.

Thus, goal **t(X, Y)** is true on this program. On the other hand, let us consider the program obtained from the previous program by replacing the third clause with

t :– true | true, p(X, Y), q(X, Y).

In this case, when goal **t** is invoked then goal clause **p(X, Y), q(X, Y)** is invoked and suspends, it cannot proceed with the computation whatever process runs in parallel with **t**. Goals such as **t** are false on the semantics presented here.

4. Fixpoint semantics

This section discusses the fixpoint characterization of the semantics of programs.

It is easy to show that the set of all interpretations *IP* defined from *I/O-hist* is a complete lattice with a partial order of set inclusion. The maximum element is *I/O-nist* and the minimum element is $\emptyset$.

Definition 4.1. Let D be a program. $\Phi_D : IP \rightarrow IP$ is the function defined as follows:

$$
\begin{aligned}
\Phi_D(S) = S \cap \{t \mid & \text{ each element of } t \text{ has the form of} \\
& H :- \{\langle\{U_{g1}, \ldots, U_{gm},\} \mid uni(X_1, \tau_1)\rangle, \ldots, \\
& \langle\{U_{g1}, \ldots, U_{gm},\} \mid uni(X_h, \tau_h)\rangle\} \cup \\
& \quad ((GU_1 \| \cdots \| GU_k) \bowtie (\{U_{g1}, \ldots, U_{gm}\}, \\
& \quad \{X_1 = \tau_1, \ldots, \\
& \quad X_h = \tau_h\}))\downarrow Var(H) \text{ for some clause in } D: \\
& \quad H :- U_{g1}, \ldots, U_{gm} \mid X_1 = \tau_1, \ldots, X_h = \tau_h, B_1, \ldots, B_k \\
& \text{where } GU_i \text{ is the body part of a element of} \\
& \text{the trace } t (\in S) \text{ of } \sigma B_i. \\
& \sigma \text{ is an } \omega\text{-substitution such that} \\
& \quad \sigma = \{U_{g1}, \ldots, U_{gm}, X_1 = \tau_1, \ldots, X_h = \tau_h\} \cup \sigma' \\
& \text{for some } \sigma', \text{ restricted to} \\
& \quad Var(H) \cup \{X_1, \ldots, X_h\}, \text{ and} \\
& \theta \subset \sigma \text{ for all} \\
& \quad \langle\theta \mid U\rangle \in (GU_1 \| \cdots \| GU_k) \bowtie (\{U_{g1}, \ldots, U_{gm}\}, \\
& \quad \{X_1 = \tau_1, \ldots, X_h = \tau_h\})\}.
\end{aligned}
$$

For a chain $S_i : S_0 \supset S_1 \supset S_2 \supset \cdots$, the greatest lower bound of S_i is denoted $\bigcap \{S_i \mid 0 \leq i\}$.

Definition 4.2. Let L be a complete lattice. A function $f : L \rightarrow L$ is *ω-continuous from below*, if for any chain $S_i : S_0 \supset S_1 \supset S_2 \supset \cdots$,

$$\bigcap \{f(S_i) \mid 0 \leq i\} = f(\bigcap \{S_i \mid 0 \leq i\}).$$

It is well known that if f is ω-continuous from below then f is monotone, that is if $S_1 \supset S_2$, then $f(S_1) \supset f(S_2)$. The following two propositions are well known (see [13]).

Proposition 4.3. *Let L be a complete lattice with the maximum element, $\top$. If $f: L \to L$ is ω-continuous from below, then f has the greatest fixpoint* $\mathrm{gfp}f$ *and*

$$\mathrm{gfp}f = \bigcap \{f^n(\top) \mid n \geq 0\}$$

where $f^0(X) = X$, $f^{n+1}(X) = f(f^n(X))$.

Proposition 4.4. *If f is a monotone function, then*

$$\mathrm{gfp}f = \bigcup \{X \mid f(X) \supset X\}$$

where $\bigcup S$ is the least upper bound of S.

The following properties are easy to show from the definitions.

Proposition 4.5. *Φ_D is ω-continuous from below.*

Proof. (1) $\Phi_D(\bigcap \{S_j \mid 0 \leq j\}) \subset \bigcap \{\Phi_D(S_j) \mid 0 \leq j\}$: For any $t \in \Phi_D(\bigcap S_j)$, there exists a clause

$$H :- U_{g1}, \ldots, U_{gm} \mid X_1 = \tau_1, \ldots, X_h = \tau_h, \quad B_1, \ldots, B_k \in D$$

An element of t has the form

$$H :- \{\langle \{U_{g1}, \ldots, U_{gm}\} \mid uni(X_1, \tau_1)\rangle, \ldots, \langle \{U_{g1}, \ldots, U_{gm}\} \mid uni(X_h, \tau_h)\rangle\} \cup ((GU_1 \| \cdots \| GU_k) \bowtie (\{U_{g1}, \ldots, U_{gm}\}, \{X_1 = \tau_1, \ldots, X_h = \tau_h\})\downarrow Var(H)$$

where GU_i is the body part of an element of a trace $t_i (\in \bigcap \{S_j \mid 0 \leq j\})$ of σB_i and σ is an ω-substitution which satisfies the condition in Definition 4.1. From $t_i \in \bigcap \{S_j \mid 0 \leq j\}$, $t_i \in S_j$ for all j. (Note that σ can be chosen independently from j.) Thus for all j, $t \in \Phi_D(S_j)$ from the definition of Φ_D. Since, $\bigcap \{\Phi_D(S_j) \mid 0 \leq j\}$ is the greatest lower bound of $\{\Phi(S_i) \mid 0 \leq i\}$, then

$$t \in \bigcap \{\Phi_D(S_j) \mid 0 \leq j\}.$$

(2) $\Phi_D(\bigcap \{S_j \mid 0 \leq j\}) \supset \bigcap \{\Phi_D(S_j) \mid 0 \leq j\}$: Assume $t \in \bigcap \{\Phi_D(S_j) \mid 0 \leq j\}$. For any j, $t \in \Phi_D(S_j)$. An element of t has the form of

$$H :- \{\langle \{U_{g1}, \ldots, U_{gm}\} \mid uni(X_1, \tau_1)\rangle, \ldots, \langle \{U_{g1}, \ldots, U_{gm}\} \mid uni(X_h, \tau_h)\rangle\} \cup ((GU_1 \| \cdots \| GU_k) \bowtie (\{U_{g1}, \ldots, U_{gm}\}, \{X_1 = \tau_1, \ldots, X_h = \tau_h\}))\downarrow Var(H)$$

where GU_i is the body part of an element of a trace $t_i (\in \bigcap \{S_j \mid 0 \leq j\})$ of σB_i and σ is an ω-substitution which satisfies the condition in Definition 4.1. For any j, since $t_i \in S_j$, then $t_i \in \bigcap \{S_j \mid 0 \leq j\}$ for each i. Thus, $t \in \Phi_D(\bigcap \{S_j \mid 0 \leq j\})$ from the definition of Φ_D. □

Proposition 4.6. *$\Phi_D(I) \supset I$ if and only if I is a model of D.*

Proof. (If part): Let I be a model of D and $t \in I$. From Definition 3.15, there exists a clause

$$H :- U_{g1}, \ldots, U_{gm} \mid X_1 = \tau_1, \ldots, X_h = \tau_h, \quad B_1, \ldots, B_k \in D$$

and t has the form

$$H :- \{\langle\{U_{g1}, \ldots, U_{gm}\} \mid uni(X_1, \tau_1)\rangle, \ldots, \langle\{U_{g1}, \ldots, U_{gm}\} \mid uni(X_h, \tau_h)\rangle\} \cup ((GU_1 \| \cdots \| GU_k) \bowtie (\{U_{g1}, \ldots, U_{gm}\}, \{X_1 = \tau_1, \ldots, X_h = \tau_h\}))\downarrow Var(H)$$

where GU_i is the body part of an element of a trace $t_i \in I$ of σB_i and σ is an ω-substitution which satisfies the condition in Definition 3.15 (and the condition in Definition 4.1). From the definition of Φ_D, $t \in \Phi_D(I)$.

(only if part): Assume $\Phi_D(I) \supset I$, in other words for any $t \in I$, $t \in \Phi_D(I)$. From the definition of $\Phi_D(I)$,

$t \in \{t \mid$ there exists a clause
$H :- U_{g1}, \ldots, U_{gm} \mid X_1 = \tau_1, \ldots, X_h = \tau_h, \quad B_1, \ldots, B_k \in D$,
each element of t has the form
$H :- \{\langle\{U_{g1}, \ldots, U_{gm},\} \mid uni(X_1, \tau_1)\rangle, \ldots, \langle\{U_{g1}, \ldots, U_{gm},\} \mid uni(X_h, \tau_h)\rangle\} \cup ((GU_1 \| \cdots \| GU_k) \bowtie (\{U_{g1}, \ldots, U_{gm}\}, \{X_1 = \tau_1, \ldots, X_h = \tau_h\}))\downarrow Var(H)$
where GU_i is the body part of some instance of a trace $t_i (\in I)$
of the goal σB_i, σ is an ω-substitution such that
$\sigma = \{U_{g1}, \ldots, U_{gm}\} \cup \{X_1 = \tau_1, \ldots, X_h = \tau_h\} \cup \sigma'$,
which is restricted to $Var(H) \cup \{X_1, \ldots, X_n\}$ and
$\forall \langle\theta \mid U\rangle \in (GU_1 \| \cdots \| GU_k) \bowtie (\{U_{g1}, \ldots, U_{gm}\}, \{X_1 = \tau_1, \ldots, X_h = \tau_h\}), \ \theta \subset \sigma\}$.

This is the definition of the model of D. □

Thus, we derive the following theorem.

Theorem 4.7. *Let D be a program and M_D be the maximum model of D. Then*

$$M_D = \bigcap \{\Phi_D^n(I/O\text{-}hist) \mid n \geq 0\}.$$

5. Conclusion

This paper presented a new declarative semantics for a subset of Flat GHC programs based on the maximum model. Using the semantics, the solutions of programs which contain perpetual processes controlled by guard commit mechanisms can be characterized as the logical consequence of the programs.

The semantics presented here is a kind of success set semantics. Thus, it is enough to discuss the results of normal computation. However, a true goal clause on this semantics is a goal clause which can run normally. This does not mean that a true

goal clause always runs normally. For example, if a subgoal of the clause commits to a different clause, then it can fail or deadlock. GHC is a *don't care non-deterministic language.* Thus, a method to characterize the set of goal clauses which run in any case is also expected. We reported on failure/deadlock set of GHC programs in [12].

Acknowledgment

I would like to thank Dr. Furukawa, the researchers of the First Laboratory of ICOT and Professor Levi of Università di Pisa for their helpful discussions.

References

[1] K. Apt and M.H. Van Emden, Contributions to the theory of logic programming, *J. Assoc. Comput. Mach.* **29** (1982).

[2] J.W. de Bakker and J.N. Kok, Uniform abstraction, atomicity and contractions in the comparative semantics of concurrent Prolog, in: *Proc. Internat. Conf. on Fifth Generation Computer Systems*, Tokyo (1988) 347-355.

[3] K.L. Clark and S. Gregory, PARLOG: parallel programming in logic, *ACM Trans. Programming Language Systems* **86** (1986).

[4] M. Falaschi, G. Levi, M. Martelli and C. Palamidessi, A more general declarative semantics for logic programming languages, Dipartimento di Informatica, Università di Pisa, Italy, Tech. Report, January 1988.

[5] M. Falaschi and G. Levi, Operational and fixpoint semantics of a class of commited-choice logic languages, Dipartimento di Informatica, Università di Pisa, Italy, Techn. Report, January 1988.

[6] H. Gairman, M.J. Maher and E. Shapiro, Existential constraints, reactive behaviors and fully abstract compositional semantics of concurrent logic programs, to appear in: *Proc. North American Conf. on Logic Programming* (1989) to appear.

[7] G. Levi and C. Palamidessi, An approach to the declarative semantics of synchronization in logic languages, in: *Proc. Internat. Conf. on Logic Programming* **87** (1987).

[8] G. Levi, A new declarative semantics of Flat Guarded Horn Clauses, ICOT Technical Report, TR-345, 1988.

[9] J.W. Lloyd, *Foundations of Logic Programming* (Springer, Berlin, 1984).

[10] M.J. Maher, Logic semantics for a class of committed-choice programs, in: *Proc. Internat. Conf. on Logic Programming* **87** (1987).

[11] M. Murakami, A declarative semantics of parallel logic programs with perpetual processes, in: *Proc. Internat. Conf. on Fifth Generation Computer Systems*, Tokyo (1988).

[12] M. Murakami, A declarative semantics of parallel logic programs based on failure/deadlock set, ICOT Technical Memo, TM-602, 1988.

[13] D. Park, Fixpoint induction and proofs of program properties, *Machine Intelligence* **5** (Edinburgh University Press, Edinburgh, 1969).

[14] Y. Sakakibara, A fixpoint characterization of stream parallelism in logic programs, in: *Proc. 2nd Conf. JSSST* (1985) in Japanese.

[15] V.A. Saraswat, *Partial Correctness Semantics for CP*[↓, |, &], Lecture Notes in Computer Science **206** (Springer, Berlin, 1985).

[16] V.A. Saraswat, The Concurrent logic programming CP: definition and operational semantics, in: *Proc. ACM Symp. on Principles of Programming Languages* (1987).

[17] V.A. Saraswat, GHC: operational semantics, problems and relationship with CP(↓, |), in: *IEEE Internat. Symp. on Logic Programming*, San Francisco (1987) 347-358.

[18] V.A. Saraswat, Concurrent constraint programming languages, Ph.D. thesis of Computer Science Department, Carnegie Mellon University, 1989.

[19] E.Y. Shapiro, *Concurrent Prolog: A Progress Report*, Lecture Notes in Computer Science **232** (Springer, Berlin, 1986).
[20] E. Shibayama, A compositional semantics of GHC, in: *Proc. 4th Conf. JSSST* (1987).
[21] A. Takeuchi, Towards a semantic model of GHC, Tech. Rep. of IECE, COMP86-59, 1986.
[22] K. Ueda and K. Furukawa, Transformation rules for GHC programs, in: *Proc. Internat. Conf. on Fifth Generation Computer Systems*, Tokyo (1988) 582-591.
[23] K. Ueda, Guarded Horn Clauses (MIT Press, Cambridge, MA) to appear.